# Dotnet build settings

This document describes the `dotnet_pkg_info` tool and how to include the .NET package build settings in the SWAMP assessment framework's `package.conf` file.

# 1. `dotnet_pkg_info` Tool

## 1.1. Introduction

Microsoft *.NET core* is a cross-platform framework that can be used to build ASP.NET Core web applications, command-line applications, libraries, and universal windows platform applications. The .NET applications are written in *C#, F#, or Visual Basic* programming languages. The .NET applications typically use MSBuild as the build system. The *MSBuild* files, commonly referred to as *project files* adhere to MSBuild XML schema. These *project files* end with extensions `.csproj`, `.vbproj` or `.fsproj`. .NET packages that use *Microsoft Visual Studios* for development have one or more Visual Studios Solution file (`.sln` extension) in their package directories. A visual studios *solution* files for a package is a text file that has information such as *project* file paths and *build configurations* (`Debug` or `Release`) for the modules in the project.

The `dotnet_pkg_info`, command line tool returns build settings about a .NET package in a `json` or `text` format. The `dotnet_pkg_info` tool is intended to be used by SWAMP UI team to help users add .NET packages to SWAMP. Code from `dotnet_pkg_info` tool is also be used in the SWAMP assessment to get build settings of a package during the build.

Functionality of `dotnet_pkg_info`:

1. List *Visual Studios Solution Files*, *Project Files*, *Target Frameworks* and *Build Configuration* in a package (Build Settings)
2. List common .NET *File Type Extensions* and related data
3. List known .NET *Target Frameworks* and related data

## 1.2. Package Build Settings

The main function of `dotnet_pkg_info` is to get build settings of a package in a `json` or `text` format. It does this when it is given arguments that are paths to Visual Studios Solution Files (file with `.sln` extension), .NET Project Files or a directories (recursively searched for solution or project files).

The .NET ecosystem lets packages to build against different libraries and APIs, some of these may only be available on Windows. In the .NET ecosystem this is referred as *targeting a framework*. When an application targets a particular framework or a set of frameworks, the *project* file for that application must define `TargetFramework` or `TargetFrameworks` attribute whose values are one or more *Target Framework Moniker (TFM)*, see Appendix A for the current list.

To get build settings of a package, execute `dotnet_pkg_info` with the path to a package directory or a solution file or a project file as an argument. If the argument is a package directory, `dotnet_pkg_info` recursively searches for *solution* files in the package. For each of the *solution files*, it lists the *project files*. And for each of the *project files*, `dotnet_pkg_info` lists the provided *target frameworks* and *build configurations*. The tool also returns the Errors and Warnings if it encounters any.

The options and arguments for `dotnet_pkg_info` are listed in the table dotnet_pkg_info Options and Arguments:

*Table 1. dotnet_pkg_info Options and Arguments:*

| Option/Argument | Description |
|---|---|
| `PACKAGE` | Path to a directory or a *solution* file or a *project* file |
| `--format FORMAT` | the values `text` or `json`. Default is `json` |
| `--no-config` | Do not display configuration information |
| `--no-framework` | Do not display target framework information |
| `--build-settings BUILD_SETTINGS` | Validates the build settings (BUILD_SETTINGS) provided as a `json` string, combines the BUILD_SETTINGS with results from getting information about the PACKAGE. |
| `--src-file-types` | list of .NET source file extensions |
| `--framework-types` | list of frameworks available on |
| `--proj-file-types` | list of .NET msbuild project file extensions |

*Example: build settings in JSON format*

```
% dotnet_pkg_info ./Identity-2.0.1
```

*Output*

```
{
  "sln_files": {
    "Identity.sln": [
      "src/AspNetCore.Identity/AspNetCore.Identity.csproj",
      "src/Extensions.Identity.Core/Extensions.Identity.Core.csproj"
    ],
    "IdentityCore.sln": [
      "src/AspNetCore.Identity/AspNetCore.Identity.csproj",
      "src/Extensions.Identity.Core/Extensions.Identity.Core.csproj",
      "src/Extensions.Identity.Stores/Extensions.Identity.Stores.csproj"
    ]
  },
  "proj_files": {
    "src/AspNetCore.Identity/AspNetCore.Identity.csproj": {
      "frameworks": ["netcoreapp2.0", "net461"],
      "configurations": ["Debug", "Release"],
      "default_framework": "netcoreapp2.0",
      "default_configuration": "Debug"
    },
    "src/Extensions.Identity.Core/Extensions.Identity.Core.csproj": {
      "frameworks": ["netstandard2.0"],
      "configurations": ["Debug", "Release"],
      "default_framework": "netstandard2.0",
      "default_configuration": "Debug"
    },
    "src/Extensions.Identity.Stores/Extensions.Identity.Stores.csproj": {
      "frameworks": ["netstandard2.0"],
      "configurations": ["Debug", "Release"],
      "default_framework": "netstandard2.0",
      "default_configuration": "Debug"
    }
  },
  "errors": {
  },
  "warnings": {
  }
}
```

*Example: build settings in text format*

```
% dotnet_pkg_info --format text ./Identity-2.0.1
```

*Output*

```
sln_files:
  Identity.sln
    src/AspNetCore.Identity/AspNetCore.Identity.csproj
    src/Extensions.Identity.Core/Extensions.Identity.Core.csproj
  IdentityCore.sln
    src/AspNetCore.Identity/AspNetCore.Identity.csproj
    src/Extensions.Identity.Core/Extensions.Identity.Core.csproj
    src/Extensions.Identity.Stores/Extensions.Identity.Stores.csproj
proj_files:
  src/AspNetCore.Identity/AspNetCore.Identity.csproj
    frameworks:
      netcoreapp2.0
      net461
    configurations:
      Debug
      Release
    default_framework:
      netcoreapp2.0
    default_configuration:
      Debug
  src/Extensions.Identity.Core/Extensions.Identity.Core.csproj
    frameworks:
      netstandard2.0
    configurations:
      Debug
      Release
    default_framework:
      netstandard2.0
    default_configuration:
      Debug
  src/Extensions.Identity.Stores/Extensions.Identity.Stores.csproj
    frameworks:
      netstandard2.0
    configurations:
      Debug
      Release
    default_framework:
      netstandard2.0
    default_configuration:
      Debug
errors:
warnings:
```

**NOTE** To get build settings without *Build Configuration* and *Target Framework* information, use `--no-config` and `--no-framework` option to the `dotnet_pkg_info` command.

### 1.2.1. For packages without the solution files

If a package does not have a *solution file* any where in the package directory, the tool recursively searches the package for *project files*. It lists the *project files* along with *target frameworks* mentioned in the *project files*. Note that *build configuration* information won't be available in this case as *build configuration* is provided in the *solution files*.

*Example: build settings with no solution files*

```
% dotnet_pkg_info ./Identity-2.0.1
```

*Output*

```
{
  "sln_files": {
  },
  "proj_files": {
    "src/AspNetCore.Identity/AspNetCore.Identity.csproj": {
      "frameworks": ["netcoreapp2.0", "net461"],
      "default_framework": "netcoreapp2.0",
    },
    "src/Extensions.Identity.Core/Extensions.Identity.Core.csproj": {
      "frameworks": ["netstandard2.0"],
      "default_framework": "netstandard2.0",
    },
    "src/Extensions.Identity.Stores/Extensions.Identity.Stores.csproj": {
      "frameworks": ["netstandard2.0"],
      "default_framework": "netstandard2.0",
    }
  },
  "errors": {
  },
  "warnings": {
  }
}
```

## 1.3. Validate and Merge a Package's Build Settings

If there is an existing .NET build settings (see section SWAMP Assessment Framework `package.conf` .NET Settings), then that same data can be passed as an additional option to both verify if the data is still valid and to populate the resulting build settings with these values. This can be used to populate the selection in the UI displayed to the users to set current values.

Given .NET package build settings in `json` format, the `dotnet_pkg_info` tool verifies that the build settings are correct or not for a given package. i.e if the given *solution* and *project* files are present in the package, *target frameworks* and *build configuration* are still valid for the project files. It merges the `json` results with the actually reading the build settings

The resulting data structure is same as that described in Package Build Settings with the optional

addition of two properties `framework` and `configuration` to the project attributes. It is the framework and configuration values that are validated and preserved. All other data is regenerated from the *solution* and *project* files.

The command returns an error if any *solution* or *project* files, or any `framework` or `configuration` attributes in the provided `BUILD_SETTINGS` are not present in the `PACKAGE`. The format for the error message will as described in the Errors and Warnings

*Example:*

```
dotnet_pkg_info ./Identity-2.0.1 --build-settings '{
  "sln_files": {
    "Identity.sln": [
      "src/AspNetCore.Identity/AspNetCore.Identity.csproj",
      "src/Extensions.Identity.Core/Extensions.Identity.Core.csproj"
    ],
  },
  "proj_files": {
    "src/AspNetCore.Identity/AspNetCore.Identity.csproj": {
      "framework": "netcoreapp2.0",
      "configuration": "Debug",
    },
    "src/Extensions.Identity.Core/Extensions.Identity.Core.csproj": {
      "framework": "netstandard2.0",
      "configuration": "Debug"
    },
  }
}'
```

*Output*


```
{
  "sln_files": {
    "Identity.sln": [
      "src/AspNetCore.Identity/AspNetCore.Identity.csproj",
      "src/Extensions.Identity.Core/Extensions.Identity.Core.csproj"
    ],
    "IdentityCore.sln": [
      "src/AspNetCore.Identity/AspNetCore.Identity.csproj",
      "src/Extensions.Identity.Core/Extensions.Identity.Core.csproj",
      "src/Extensions.Identity.Stores/Extensions.Identity.Stores.csproj"
    ]
  },
  "proj_files": {
    "src/AspNetCore.Identity/AspNetCore.Identity.csproj": {
      "frameworks": ["netcoreapp2.0", "net461"],
      "configurations": ["Debug", "Release"],
      "default_framework": "netcoreapp2.0",
      "default_configuration": "Debug",
      "framework": "netcoreapp2.0",
      "configuration": "Debug",
    },
    "src/Extensions.Identity.Core/Extensions.Identity.Core.csproj": {
      "frameworks": ["netstandard2.0"],
      "configurations": ["Debug", "Release"],
      "default_framework": "netstandard2.0",
      "default_configuration": "Debug",
      "framework": "netstandard2.0",
      "configuration": "Debug"
    },
    "src/Extensions.Identity.Stores/Extensions.Identity.Stores.csproj": {
      "frameworks": ["netstandard2.0"],
      "configurations": ["Debug", "Release"],
      "default_framework": "netstandard2.0",
      "default_configuration": "Debug"
    }
  },
  "errors": {
  },
  "warnings": {
  }
}
```


## 1.4. Target Frameworks

To display *target frameworks* available on a SWAMP platform, use `--framework-types` option with `dotnet_pkg_info` tool.

*Example*

```
dotnet_pkg_info --framework-types
```

Output

```
{
  ".NET Standard": {
    "tf_moniker" : [
      "netstandard1.0",
      "netstandard1.1",
      "netstandard1.2",
      "netstandard1.3",
      "netstandard1.4",
      "netstandard1.5",
      "netstandard1.6",
      "netstandard2.0",
      "netcoreapp1.0",
      "netcoreapp1.1",
      "netcoreapp2.0",
      "netcoreapp2.1"
    ],
    "windows_only": false
  },
  ".NET Core" : {
    "tf_moniker" : [
      "netcoreapp1.0",
      "netcoreapp1.1",
      "netcoreapp2.0",
      "netcoreapp2.1"
     ],
     "windows_only": false
  },
  ".NET Framework" : {
    "tf_moniker" : [
      "net11",
      "net20",
      "net35",
      "net40",
      "net403",
      "net45",
      "net451",
      "net452",
      "net46",
      "net461",
      "net462",
      "net47",
      "net471",
      "net472"
    ],
```

```
    "windows_only": true
  },
  "Windows Store": {
    "tf_moniker" : [
      "netcore [netcore45]",
      "netcore45 [win] [win8]",
      "netcore451 [win81]"
    ],
    "windows_only": true
  },
  ".NET Micro Framework": {
    "tf_moniker" : [
      "netmf"
    ],
    "windows_only": true
  },
  "Silverlight": {
    "tf_moniker" : [
      "sl4",
      "sl5"
    ],
    "windows_only": true
  },
  "Windows Phone": {
    "tf_moniker" : [
        "wp [wp7]",
        "wp7",
        "wp75",
        "wp8",
        "wp81",
        "wpa81"
     ],
     "windows_only": true
  },
  "Universal Windows Platform": {
    "tf_moniker" : [
      "uap",
      "uap10.0"
    ],
    "windows_only": false
  }
}
```

# 1.5. Show Source .NET File Extensions

Lists the .NET source file extensions and types.

*Example*

```
% dotnet_pkg_info --src-file-types
```

Output

```
{
  ".cs": {
    "description": "C# source files",
    "windows_only": false
  },
  ".vb": {
    "description": "Visual Basics source files",
    "windows_only": true
  },
  ".fs": {
    "description": "F# source files",
    "windows_only": true
  }
}
```

## 1.6. Show .NET Project File Extensions

Lists the .NET project file extensions

```
% dotnet_pkg_info --project-file-types
```

Output

```
{
  ".csproj": {
    "description": "csharp project file"
  },
  ".vbproj": {
    "description": "Visual Basics project files"
  },
  ".fsproj": {
    "description": "fsharp project file"
  }
}
```

## 1.7. Errors and Warnings

### 1.7.1. Error Messages and Exit Status returned by `dotnet_pkg_info`

If `dotnet_pkg_info` encounters, it returns a `json` data structure that contains an array of *error properties* , each with an *error message*, *error code*, and the *file* that is cause of the error. The *error message* format and *error codes* are given in the table dotnet_pkg_info error codes.

The format for the `json` data structure is as follows:

```
{
  errors: [
    {
      "message": "<message description>",
      "code" : "<error code>"
      "file" : "<path to the file that is the cause of the error"
    },
    {
      ...
    },
    ...
  ]
}
```

*Table 2. dotnet_pkg_info error codes*

| Error Code | Exit Status | Message Format | Description |
|---|---|---|---|
| SUCCESS | 0 | | Success |
| INVALID_PACKAGE | 1 | No solution or project files found in the directory: <directory path> , | Invalid .NET package, if the package directory does not contain *solution* or *project* files |
| INVALID_SLN_FILE | 2 | Invalid *solution* file: <path> | Invalid *solution* file, not meeting the specification https://docs.microsoft.com/en-us/visualstudio/extensibility/internals/solution-dot-sln-file |
| PROJECT_FILE_NOT_FOUND | 3 | project file in the solution file not found: | Project file listed in the solution file not found |
| INVALID_PROJECT_FILE | 4 | Invalid project file: <path> | Invalid <project>_ file, not meeting the specification https://docs.microsoft.com/en-us/visualstudio/msbuild/msbuild-project-file-schema-reference |

| Error Code | Exit Status | Message Format | Description |
|---|---|---|---|
| INVALID_TARGET_FRAMEWORK | 5 | Invalid target framework: <target framework> | Invalid *target framework*. If the *target framework* specified in the package is not in the list |
| INVALID_BUILD_CONFIGURATION | 6 | Invalid build configuration: <build configuration> | https://docs.microsoft.com/en-us/dotnet/standard/frameworks |
| INVALID_FILE_EXTENSION | 7 | Invalid .NET file extension: <path> | Invalid .NET file extension |
| FILE_PERMISSION_ERROR | 8 | File Permission error | If solution, project or directory does not have read permission |

### 1.7.2. Warnings Messages returned by `dotnet_pkg_info`

If `dotnet_pkg_info` needs to convey warnings or other messages, it uses a property called warnings, which has the same format as the error property.

The format for the `json` data structure is as follows:

```
{
  "warnings": [
    {
      "message": "<message description>",
      "code" : "<warning code>"
      "file" : "<path to the file that is the cause of the error"
    },
    {
      ...
    },
    ...
  ]
}
```

*Table 3. dotnet_pkg_info warning codes*

| Warning Code | Message Format | Description |
|---|---|---|
| REQUIRES_WINDOWS | Project requires windows to build: <project path> | If the Project uses API that are Windows only. |

# 2. SWAMP Assessment Framework `package.conf` .NET Settings

If a user selects a solution file, and a certain set of project files and target frameworks and build

configuration for their package. The SWAMP UI or middleware should pass the .NET build settings to the SWAMP assessment framework in a `json` format. The information in the `json` format must be assigned to the `package-dotnet-info` attribute in the `package.conf` file.

The format for the `package-dotnet-info` should be like as the `json` output produced by `dotnet_pkg_info` tool, except for the few changes. The properties `frameworks` and `configurations`, `default_framework` and `default_configuration` should not be present. The project properties may have `framework`, `configuration` and `nobuild` properties whose value as single strings as selected by the user or assigned from the default values. If a project has a `nobuild` property, the SWAMP assessment framework *does not build* that project.

*Example:*

```
{
  "sln_files": {
    "Identity.sln": [
      "src/AspNetCore.Identity/AspNetCore.Identity.csproj",
      "src/Extensions.Identity.Core/Extensions.Identity.Core.csproj"
    ],
  },
  "proj_files": {
    "src/AspNetCore.Identity/AspNetCore.Identity.csproj": {
      "framework": "netcoreapp2.0",
      "configuration": "Debug",
    },
    "src/Extensions.Identity.Core/Extensions.Identity.Core.csproj": {
      "framework": "netstandard2.0",
      "configuration": "Debug"
    },
  }
}
```

*Scenario 1:*

User selects a *solution* but does not select projects and does not configure the projects. In this case, `package-dotnet-info` can list the *solution* file with an empty list for projects. The SWAMP *assessment framework* invokes the MSBuild system with the *solution* file as the argument. i.e. all the projects in the *solution* file will be built against frameworks and default configuration provided in the *project* files for the modules.

Example:

```
{
  "sln_files": {
    "Identity.sln": []
  }
}
```

*Scenario 2:*

User selects a *solution*, and one or more projects in the *solution*, but does not select *configuration* for the projects. In this case, `package-dotnet-info` can list the *solution* file with the list of projects selected by the user. The *assessment framework* invokes the MSBuild system for each of the selected projects. The projects will be built against frameworks and default configuration provided in the selected *project* files.

Example:

```
{
  "sln_files": {
    "Identity.sln": [
      "src/AspNetCore.Identity/AspNetCore.Identity.csproj",
      "src/Extensions.Identity.Core/Extensions.Identity.Core.csproj"
    ]
  },
  "proj_files": {
    "src/AspNetCore.Identity/AspNetCore.Identity.csproj": {
    },
    "src/Extensions.Identity.Core/Extensions.Identity.Core.csproj": {
    },
  }
}
```

*Scenario 3:*

User selects a *solution*, and one or more projects in the *solution*, and also selects *frameworks* and *configuration* for the projects. The *assessment framework* invokes the MSBuild system for each of the selected projects. The projects will be built against framework and configuration selected by the user.

Example:

```
{
  "sln_files": {
    "Identity.sln": [
      "src/AspNetCore.Identity/AspNetCore.Identity.csproj",
      "src/Extensions.Identity.Core/Extensions.Identity.Core.csproj"
    ]
  },
  "proj_files": {
    "src/AspNetCore.Identity/AspNetCore.Identity.csproj": {
      "framework": "netcoreapp2.0",
      "configuration": "Debug"
    },
    "src/Extensions.Identity.Core/Extensions.Identity.Core.csproj": {
      "framework": "netstandard2.0",
      "configuration": "Debug"
    }
  }
}
```

*Scenario 4:*

The package does not have a *solution* file, but there are one or more projects is the package, and user selects the build settings for the projects.

Example:

```
{
  "proj_files": {
    "src/AspNetCore.Identity/AspNetCore.Identity.csproj": {
      "framework": "netcoreapp2.0",
      "configuration": "Debug"
    },
    "src/Extensions.Identity.Core/Extensions.Identity.Core.csproj": {
      "framework": "netstandard2.0",
      "configuration": "Debug",
    }
  }
}
```

*Scenario 5:*

The package does not have a *solution* file, but there are one or more projects is the package, and at-least one project is marked `nobuild` by the user.

Example:

```
{
  "proj_files": {
    "src/AspNetCore.Identity/AspNetCore.Identity.csproj": {
      "framework": "netcoreapp2.0",
      "configuration": "Debug"
    },
    "src/Extensions.Identity.Core/Extensions.Identity.Core.csproj": {
      "framework": "netstandard2.0",
      "configuration": "Debug",
      "nobuild": true
    }
  }
}
```

# Appendix A: .Valid Target Framework and Target Framework Moniker

*Table 4. Valid Target Framework and Target Framework Moniker*

| Target Framework | Target Framework Moniker | Windows Only |
|---|---|---|
| .NET Standard | netstandard1.0<br>netstandard1.1<br>netstandard1.2<br>netstandard1.3<br>netstandard1.4<br>netstandard1.5<br>netstandard1.6<br>netstandard2.0 | False |
| .NET Core | netcoreapp1.0<br>netcoreapp1.1<br>netcoreapp2.0<br>netcoreapp2.1 | False |
| .NET Framework | net11<br>net20<br>net35<br>net40<br>net403<br>net45<br>net451<br>net452<br>net46<br>net461<br>net462<br>net47<br>net471<br>net472 | True |
| Windows Store | netcore [netcore45]<br>netcore45 [win] [win8]<br>netcore451 [win81] | True |
| .NET Micro Framework | netmf | True |

| Target Framework | Target Framework Moniker | Windows Only |
|---|---|---|
| Silverlight | sl4<br>sl5 | True |
| Windows Phone | wp [wp7]<br>wp7<br>wp75<br>wp8<br>wp81<br>wpa81 | True |
| Universal Windows Platform | uap [uap10.0]<br>uap10.0 [win10]<br>[netcore50] | False |

This list may be change, refer to [https://docs.com/en-us/dotnet/standard/frameworks] for the up-to date list.